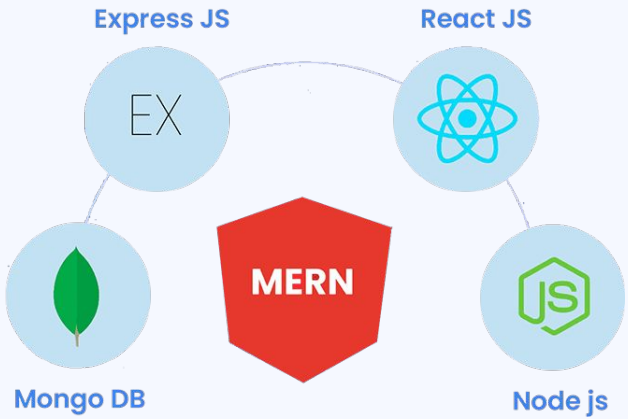


Full Stack Development Using MERN

Day 01 - 07th April 2024

Lecturer Details:

Chanaka Wickramasinghe
cmw@ucsc.cmb.ac.lk
0771570227

Anushka Vithanage
dad@ucsc.cmb.ac.lk
071 527 9016

Prameeth Maduwantha
bap@ucsc.cmb.ac.lk
077 288 8098

Course Outline:

- **08 Days** covering **60 Hours** of coursework within the class.
  - Day 01, 02, 03 - React
  - Day 04 - Node
  - Day 05 - Express
  - Day 06 - MongoDB
  - Day 07 - CRUD
  - Day 08 - Presentations and Additional Knowledge

Important Notes:

- 80% Attendance (Administration Rule)
  I.e. - you can miss only 1 class
  We need you to cover all the content (University Perspective)

- Starting Time - 8.30 AM
- Ending Time - 5.30 PM

- Morning Break around 10.30 AM (30 Mins)
- Lunch Break around 1 PM (45 Mins)
- Evening Break (5 Mins)

- Final Evaluation - Completion of Individual Project
- For Certification - Project + Attendance

Lecture 01

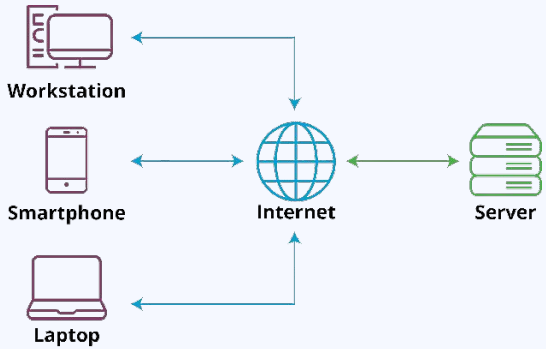
Introduction to MERN Stack Development

Client - Server Architecture

- Client requests data or services, and the server processes the request and returns the desired output.
- Key Components:
  - Client:
    - User's interface for interacting with the server.
    - Sends requests and receives responses.
      - Examples: Web browsers
  - Server:
    - Stores and manages resources.
    - Processes client requests and sends back the appropriate data.
      - Examples: Web servers

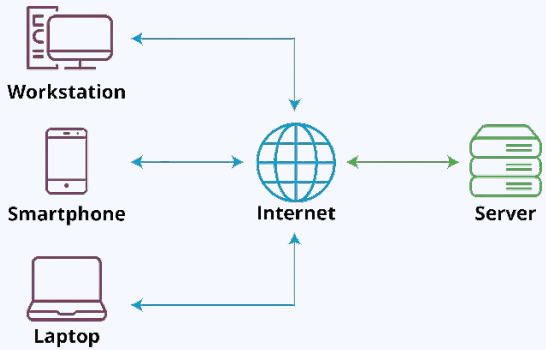
Front-end Development

- The process of creating and maintaining websites or web applications on the internet like User Interfaces.
- Components:
  - Frontend (Client-Side):
    - The visual interface that users interact with
      - Languages: HTML, CSS, JavaScript.
      - Tasks: Layout design, interactivity, user experience.



Back-end development

- Server-side development of web applications or websites
- Components:
  - Backend (Server-Side):
    - The underlying engine that processes user requests and data.
      - Languages: Java, Python, Node.js, Ruby, PHP.
      - Tasks: Database management, server configuration, data processing, API development



Full Stack development

- Development of both the front-end (client-side) and the back-end (server-side) of web applications.
- Components:
  - Frontend (Client-Side):
    - Languages: HTML (for structure), CSS (for styling), and JavaScript (for interactivity).
    - Frameworks/Libraries: React, Angular, Vue.js, etc.
  - Backend (Server-Side):
    - Languages: Java, Python, Node.js, Ruby, PHP, etc.
    - Frameworks: Express.js (for Node.js), Django (for Python), Ruby on Rails (for Ruby), Spring Boot (for Java), etc.

Frameworks

- Framework is a pre-written code structures for the development
  - When you use a framework, you don't start coding from an absolute zero; instead, the framework gives you a predefined structure or "skeleton" to build upon.
  - Without frameworks you need to manually manipulate the DOM using JavaScript, handle events, and keep track of data changes.
  - Frameworks often promote coding standards and consistent methodologies.
  - Repetitive tasks handled by the framework
    - Eg: Having components
  - Easily scalable

What is a Stack?

- A set of technologies, tools, and software used to build and run a web or mobile application.
- Each layer of technology, or each component, builds on top of the one below, creating a "stack."
- Common technology stacks include:
  - LAMP: Linux, Apache, MySQL, PHP/Perl/Python
  - MEAN: MongoDB, Express.js, AngularJS, Node.js
  - MERN: MongoDB, Express.js, React.js, Node.js

Components of Full Stack Development

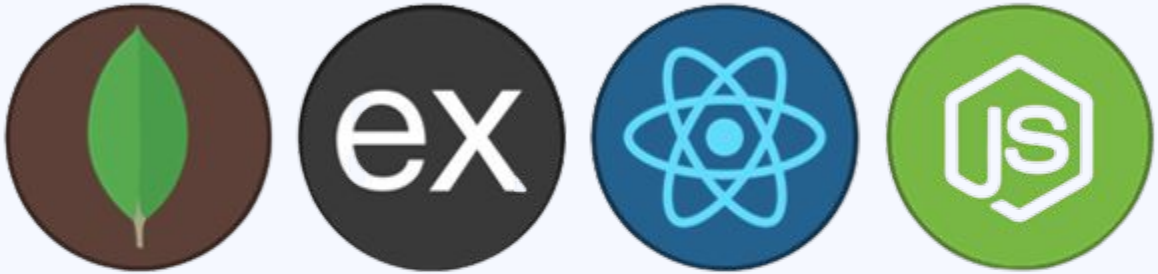
- **Front-end (Client-Side):** What the user interacts with directly.
- **Languages:** HTML, CSS, JavaScript.
- **Frameworks/Libraries:** React, Angular, Vue.js.
- **Back-end (Server-Side):** Where the logic, data processing, and storage occurs.
- **Languages:** JavaScript (Node.js), Python, Ruby, Java, PHP.
- **Frameworks:** Express.js, Django, Ruby on Rails, Spring.
- **Databases:** MongoDB, PostgreSQL, MySQL, SQLite.

Why Full Stack?

- **Versatility:** Familiar with both the frontend and backend development. Able to work on multiple aspects of a project.
- **Increased Career Opportunities:** Full Stack Developers are often more sought-after because of their broader skill set.
- **Comprehensive Solution Building:** Understand the bigger picture of a project, making it easier to make decisions that benefit the entire application.
- **Flexibility:** Can easily switch between frontend and backend tasks based on project requirements.

The Road Ahead

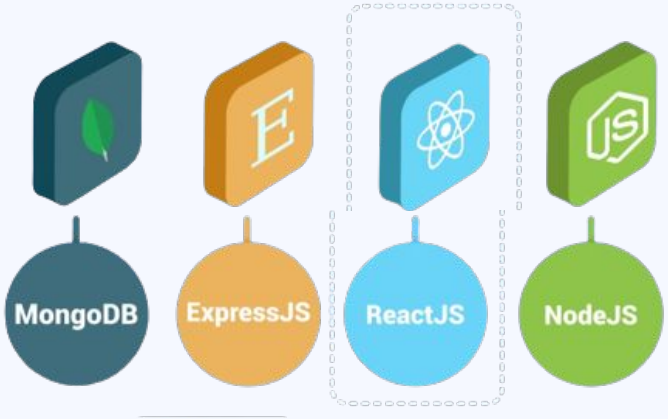
- As technology evolves, so does the role of a Full Stack Developer.
- Emergence of new tools, languages, and best practices.
- Importance of understanding user experience, security, and performance optimization.



M E R N

What is MERN Stack?

- **Acronym:** MongoDB, Express.js, React.js, Node.js.
- A full-stack JavaScript solution for building dynamic web applications.

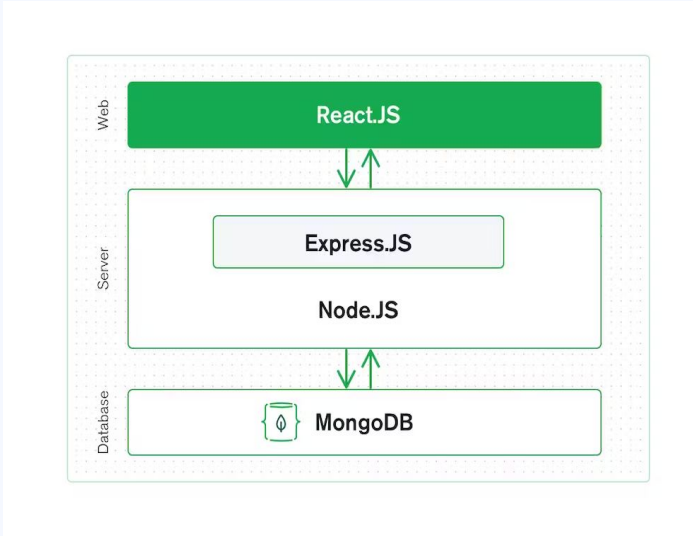


Components of MERN

- **MongoDB:** A NoSQL database that uses JSON-like documents with BSON format.
- **Express.js:** Layer for handling interactions with the database and processing client requests.
- **React.js:** A front-end library for building user interfaces.
- **Node.js:** Platform that lets you run JavaScript on the server side

Application Structure

- **Front-end:** Created using React.js, with components being the building blocks.
- **Back-end API:** Built with Express.js on a Node.js server.
- **Database:** Persistent storage with MongoDB, often interfaced using Mongoose for structured models.

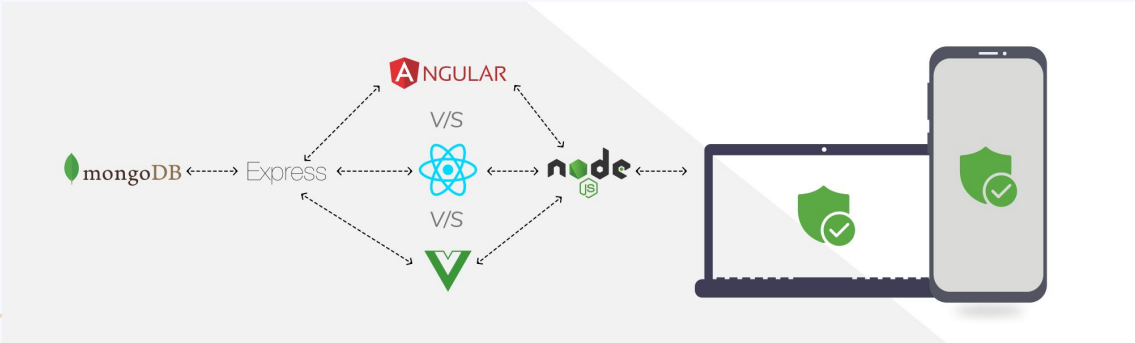


Why Choose MERN?

- **Single Language Stack:** JavaScript is used throughout the entire development, from front-end to back-end.
- **Flexibility:** Especially with MongoDB's schema-less design.
- **Performance:** React's Virtual DOM ensures that user interfaces are highly responsive and update swiftly.
- **Community Support:** Strong community, numerous libraries, and middleware.

MERN's Place in Today:

- **Modern Web Apps:** Single Page Applications (SPAs), Real-time applications.
- **Scalability:** Adaptable to a growing user base.
- **Similar Stacks:** MEAN, MEVN, etc.



Benefits of MERN:

- **Unified Language:** JavaScript Everywhere: Streamlines the development process from front-end to back-end.

  Benefit: Faster learning curve, easier debugging, and more efficient code reuse.

- **Modern & Dynamic User Experiences:** React.js Enables the creation of interactive UIs with real-time response.

  Benefit: Users get seamless, engaging, and intuitive application experiences.

Benefits of MERN:

- **Cost-Efficient & Scalable Solutions:** Single Codebase With Node.js on the backend and React on the frontend, teams can collaborate more effectively.

  Benefit: Reduce development time and cost. As the user base grows, the app can be scaled smoothly.

- **Flexibility & Freedom:** MongoDB's Schema-less Design Allows for more agile and adaptive data structures. Express.js Middleware Can be easily plugged in or changed for added functionalities.

  Benefit: Adaptable to changing project requirements, resulting in quicker iterations.

Benefits of MERN:

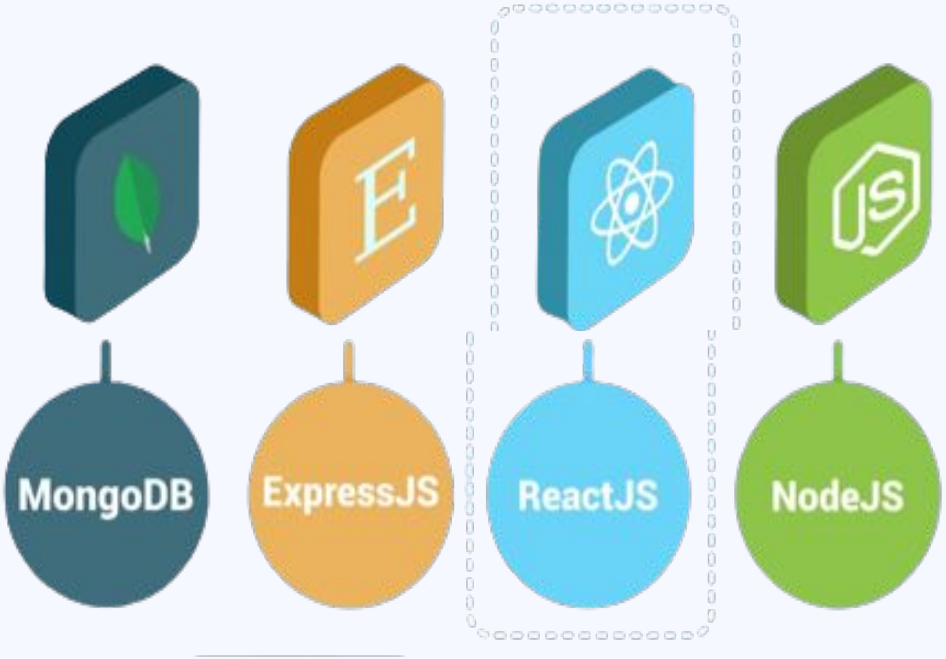
- **Strong Community & Rich Ecosystem:** Numerous libraries, middleware, and extensions available. Active support, tutorials, and frequent updates.

  Benefit: Quicker solutions to challenges, staying updated with modern tech trends.

- **Robust Performance & Security:** Non-blocking I/O of Node.js: Efficient data flow. Integrated tools in MongoDB and Express for secure data handling.

  Benefit: Build performant applications without compromising on security.

Overview of MERN Stack



MongoDB

- **Type:** NoSQL Database.
- **Usage:** Stores data for the application.
- **What is NoSQL?:** Comparison with SQL databases. NoSQL Database, Unlike SQL, data is stored in collections, not tables.
- **Collections vs Tables:** Flexibility in data storage.
- **Schema-less:** Each document can have its own unique structure.
- **JSON-like Documents:** BSON format, which supports more data types.
- **CRUD Operations:** Basic database operations in MongoDB.

MongoDB : Features

- **Schema-less:** Flexibility in data structure.
- **JSON-like documents:** Intuitive and easy to work with.
- **Scalable:** Horizontal scaling through sharding.

Express.js

- **Type:** Web Application Framework: For building web applications on top of Node.js.
- **Features:** Minimalist and fast.
- **Middleware support:** Modular and extensible.
- **Routes handling:** Manages API endpoints.

Express.js

- **Usage:** Assists in organizing the web application into an MVC architecture on the server side.
- **Routing in Express:** Handling different types of requests.
- **Middleware:** Use cases and creating custom middleware.
- **Middleware & Routing:** Provides tools to simplify tasks like user authentication, routing, etc.

React.js

What is React?

- A JavaScript library developed by Facebook.
- Primarily used for building user interfaces, especially single-page applications.
    - Landing Pages etc.
- **Type:** Front-end JavaScript Library.

React.js

Why React?

- Efficient and flexible.
- Component-based architecture.
- Reusable code components.
- Fast rendering using the Virtual DOM.

DOM

- DOM represents a document in a structured, logical manner. Think of it as turning the content of a webpage (written in HTML) into objects that can be manipulated using a programming language like JavaScript.
- Every element, attribute, and piece of text in the HTML becomes a node in this tree. For example:

```
<html>
  <head>
    <title>My Page</title>
  </head>
  <body>
    <p>Hello, World!</p>
  </body>
</html>
```

Virtual DOM

- Virtual, representation of a UI is kept in memory and synced with the "real" DOM. It's like a blueprint or plan of the changes you want to make to the real DOM.
- The real DOM is an actual representation and interface of the loaded webpage in the browser. Manipulating the real DOM directly can be slow and inefficient.
- The Virtual DOM, on the other hand, is a virtual representation of that interface. Changes made to the VDOM don't immediately affect the visible UI or the real DOM.

How the process of Virtual DOM works

- When changes occur in the UI (like user interactions or data updates), the entire UI is re-rendered in the Virtual DOM representation.
- Then, the Virtual DOM representation is compared to the previous state.
- From this difference or "diff", a minimal set of optimal changes, or patches, are calculated.
- These calculated changes are then efficiently updated or patched in the real DOM.
- This process is known as reconciliation.

React.js

Features

- JavaScript Library: For building user interfaces.
- The Virtual DOM: In-depth explanation with real-world analogy.
- Component-Based Architecture: Encourage reusable UI components.

Usage

- Builds dynamic and interactive user interfaces.
- React's Ecosystem: Tools, libraries, and communities supporting React.

Node.js

- A JavaScript runtime built on Chrome's V8 JavaScript engine.
- Allows execution of JavaScript outside of the browser, primarily on the server-side.
- Developed by Ryan Dahl in 2009.
- Type: Server-side JavaScript Runtime Environment.

Node.js

Why Node.js?

Event-driven, Non-blocking I/O Model: Efficient and lightweight, especially for data-intensive real-time applications that run across distributed devices.

Single Language Stack: Write both server-side and client-side code in JavaScript.

Expansive NPM Registry: Access to a large number of packages and modules, making development quicker and more manageable.

Node.js : Features

JavaScript Runtime: Allows you to run JavaScript on the server.

Event Loop: Handles concurrent operations without multi-threading, making it efficient.

Built-in HTTP and File System modules: Facilitates the creation of web servers and the handling of I/O operations.

Buffer: Provides a way to work with data streams directly, without conversions.

NPM (Node Package Manager): A powerful package manager to install various libraries and tools.

Cluster Module: Enables load balancing over multiple CPU cores.

Node.js

Usage:

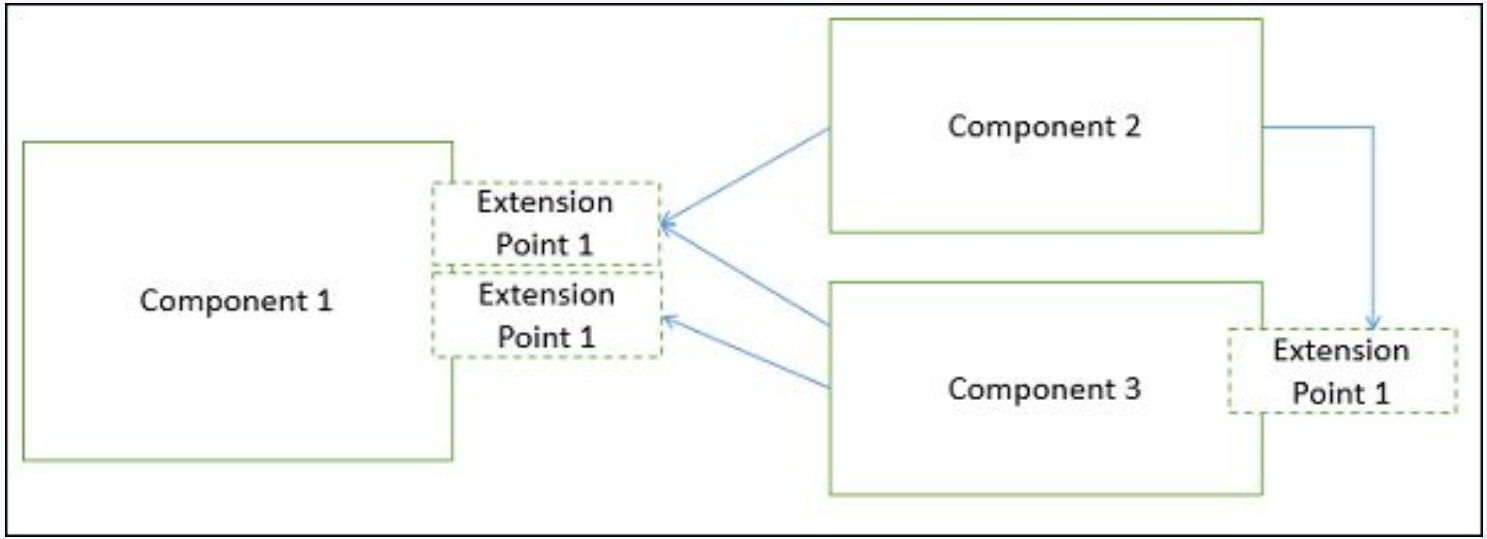
- Builds scalable network applications, especially I/O-bound applications.
- API development, real-time applications like chat applications, and collaborative tools.
- Also commonly used for scripting and automation tasks.

Node.js

Node.js Ecosystem:

- **Express.js:** A minimalist web application framework for Node.js.
- **Socket.io:** Enables real-time, bidirectional, and event-driven communication.
- **Mongoose:** A MongoDB object modeling tool.
- **Passport:** An authentication middleware for Node.js.
- **Communities and Forums:** A vibrant community, with lots of tutorials, blogs, and courses dedicated to Node.js development.

Component Based Architecture



What are Components?

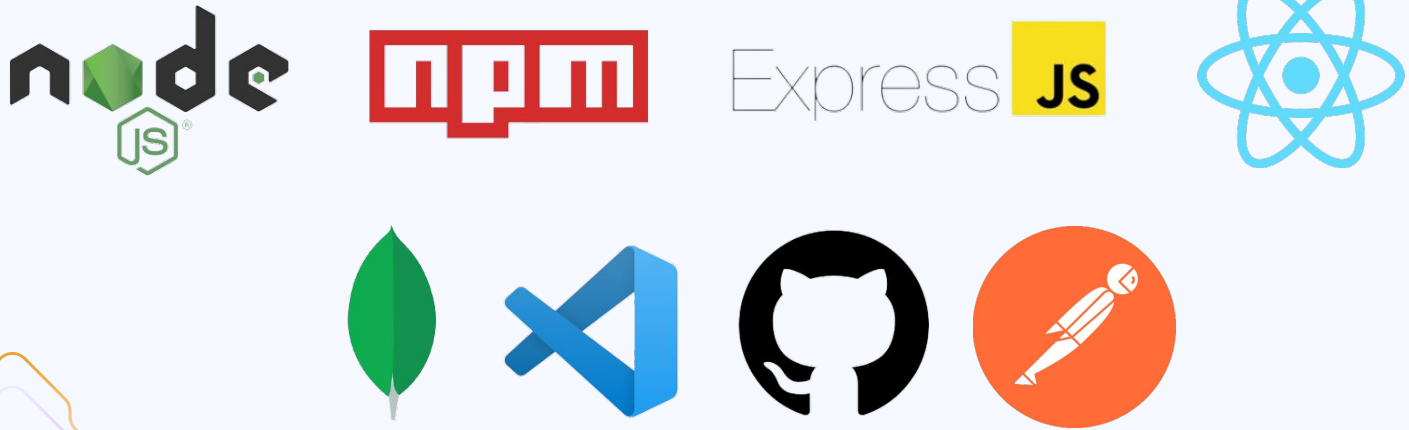
- Self-contained, reusable chunks of UI.
- Like JavaScript functions: they accept inputs (props) and return React elements describing UI.

Benefits:

- Modular development.
- Code reuse across the app.
- Easier to maintain and scale.

Format – JSX

Setting Up the Development Environment for MERN Stack Development



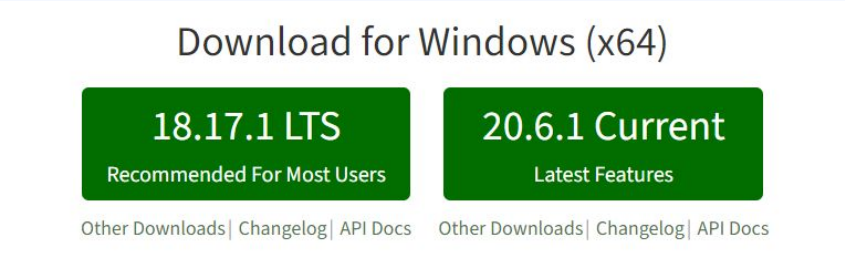
Setting Node.js and NPM

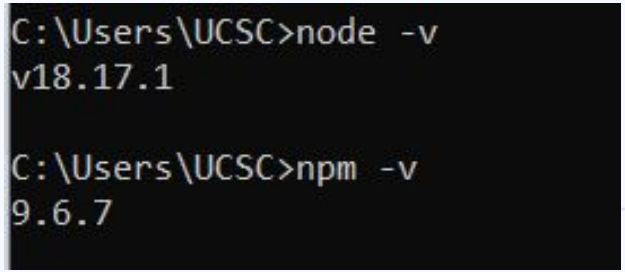
Download: Visit Node.js official website. (https://nodejs.org/en)

Install: Choose the LTS version for stability.

Verify Installation:

Run “node -v” and “npm -v” in the terminal to check versions.





Setting React.js

Create React App (CRA): A toolchain to set up a new React project.

Installation: Run “npx create-react-app my-app”

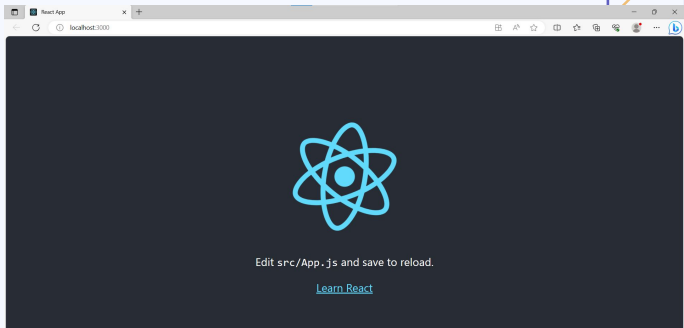
Initialize npm: Run “npm init” and “npm audit fix”

Start Development Server: cd my-app && npm start

```
C:\Users\UCSC\Desktop\mern>npx create-react-app my-app
Need to install the following packages:
create-react-app@5.0.1
Ok to proceed? (y) y
```

```
C:\Users\UCSC\Desktop\mern>cd my-app

C:\Users\UCSC\Desktop\mern\my-app>npm start
```



Syntax

JSX, or JavaScript XML, is a unique feature of React, JSX is a syntax extension for JavaScript. It's not a programming language by itself. It combines elements of both HTML and JavaScript.

JSX vs Traditional HTML

- JSX is more dynamic, allowing the embedding of JS expressions.
- HTML is static and can't have logic embedded directly.

JSX vs Javascript

- JSX is a blend of JS logic and a markup language.

Development Tools & Extensions

Integrated Development Environment (IDE):

Recommended - Visual Studio Code (VSCode).

VSCode Extensions:

- ESLint: For linting and code quality.
- ES7 React/Redux/GraphQL/React-Native snippets
- Prettier: Code formatter.
- React extensions: Reactjs code snippets, React Developer Tools.

VSCode Extensions

ESLint –

Code Quality: ESLint improves code quality by identifying and fixing issues in JavaScript code.

Code Consistency: It enforces a consistent coding style across the project.

Issue Identification: ESLint scans code and detects potential issues based on configured rules.

Customizable Rules: Rules and configurations are highly customizable to match specific coding conventions.

VSCode Extensions

ESLint –

Automatic Fixes: ESLint can automatically correct many code issues.

Integration: It seamlessly integrates into code editors, providing real-time feedback.

Configurations: ESLint rules are defined in .eslintrc files, allowing for project-specific settings.

Extensibility: It can be extended with plugins and custom rules to suit project requirements.

VSCode Extensions

React/Redux/GraphQL/React-Native snippets -

Code Snippets: Provides a library of ready-to-use code snippets for React, Redux, GraphQL, and React Native development.

React Support: Includes snippets for creating React components and common lifecycle methods, speeding up React application development.

Redux Integration: Offers snippets for creating Redux actions, reducers, and store configurations, simplifying state management in Redux.

VSCode Extensions

React/Redux/GraphQL/React-Native snippets -

Customization: Allows developers to create and customize their own snippets to match project-specific requirements and coding styles.

Improved Productivity: Enhances development productivity by reducing the need to write repetitive code and search for documentation.

Learning Aid: Serves as a learning tool with working examples and templates, helping developers understand and implement best practices for these technologies.

VSCode Extensions

Prettier –

Code Formatting: Prettier automatically formats code for consistent styling, including indentation and spacing.

Consistency: It enforces a single, opinionated code style for the entire project, reducing debates about code formatting.

Opinionated: Prettier has strict, predefined formatting rules, minimizing the need for configuration.

Language Support: It supports multiple programming languages, making it versatile for various project types.

VSCode Extensions

Prettier –

Integration: Prettier integrates with code editors and build tools to automatically format code during development.

Pre-commit Hooks: It can be set up to run as a pre-commit hook, ensuring consistent code formatting in version control.

Configurability: While opinionated, Prettier allows minimal configuration to align with project preferences.

Focus on Code Content: Prettier concentrates solely on code formatting, leaving code quality and logic to other tools and practices.

VSCode Extensions

React Extension Pack -

Code Snippets: Offers code snippets for creating React components and common React patterns, improving development speed and consistency.

Intelligent Auto-Completion: Provides smart auto-completion suggestions for React-specific code constructs, enhancing coding efficiency.

Syntax Highlighting: Ensures proper highlighting of React code elements for better code readability and error identification.

VSCode Extensions

React Extension Pack -

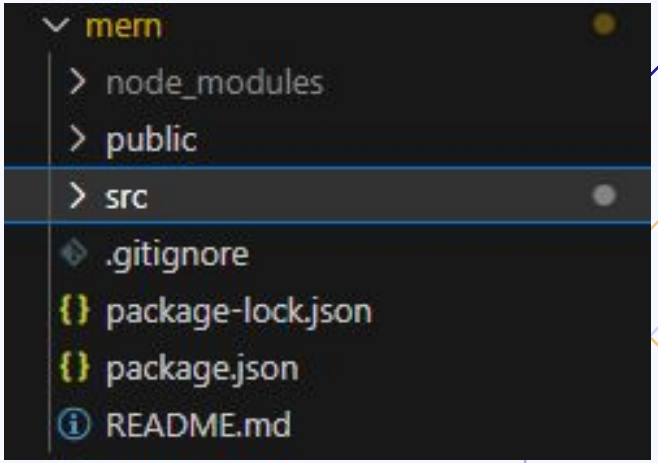
Component Tree Visualization: Visualizes the hierarchy of React components, aiding in understanding complex component structures.

Linting and Error Checking: Includes tools like ESLint and TypeScript support for static code analysis and error detection, ensuring code quality.

Extension Updates: Provides tools to keep bundled extensions up to date, ensuring access to the latest React development features and improvements.

Structure

- package.json
  - "dependencies" lists packages your project depends on.
  - "scripts" contains shortcut commands you can run with npm
- node_modules
  - This folder contains all the dependencies (libraries/packages) that your project uses. These are listed in your package.json file and are installed when you run npm install



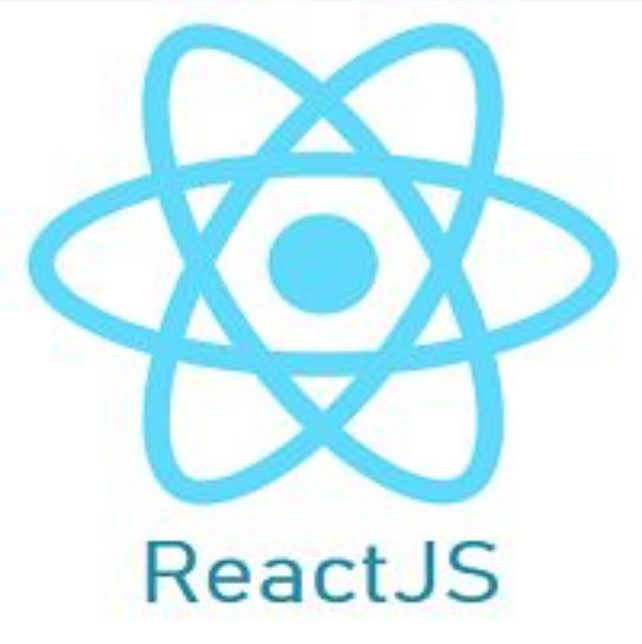
Structure

- Public - This folder has public assets and the root HTML file:
  - index.html: The main HTML file. React injects its components into the element.
  - favicon.ico: The default favicon (short for "favorite icon"). This is the small icon you see in the browser tab.
  - manifest.json: This is related to Progressive Web Apps (PWA). It provides information about the application (name, short name, icons, and more) to the browser.
- package-lock.json
  - This file ensures that the dependencies remain the same across all environments

Structure

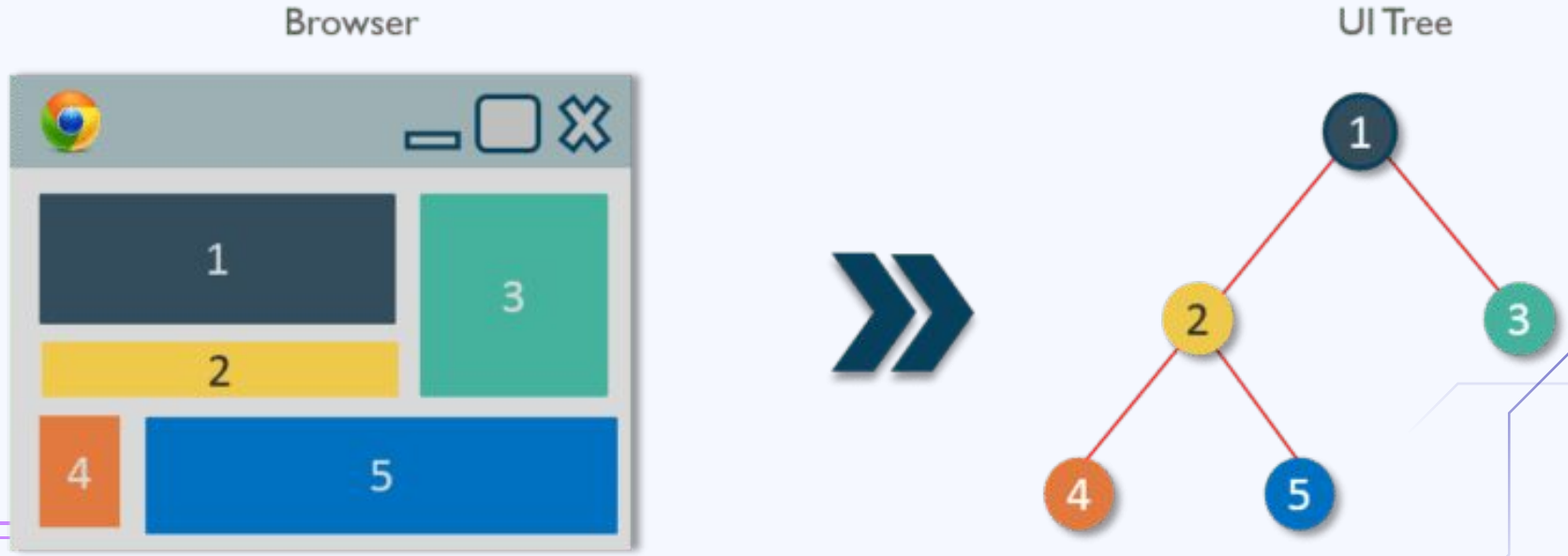
- src - The heart of your React application. All the React code resides here:
  - App.css: The default styles for the App component.
  - App.js: This is the main component file. By default, it renders a simple intro to React.
  - App.test.js: A sample test file for the App component using Jest.
  - index.css: Global styles for your application.
  - index.js: The entry point for your React application. Here the React DOM renders the App component inside the root div in index.html.
  - setupTests.js: Configuration file for Jest, a testing tool.

DEMO



Components in React

- Building blocks of your UI (User Interface). Each component is a reusable piece of UI that can be combined with other components to build complex user interfaces



Class Components

- Class components based on JavaScript ES2015. It extends Component base class from React and it gives way to the react life cycle methods and add a render function which is use for return the react elements.

```
class App extends Component {
  render() {
    return (
      <View>
        <Text>Hello, world!</Text>
      </View >
    )
  }
};
```

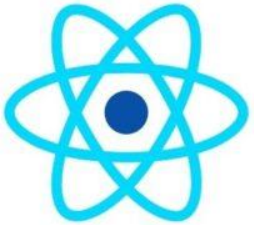
Functional Components

- Functional components based on simple or plain JavaScript. Functional components can’t maintain their own state that’s why sometimes we can say it stateless components. It is just accept the props as an argument and return the react elements.

```
function Hello(props) {
  return <h2>Hello Dear, {props.name}</h2>;
}
```

Core concepts

Of



React

Props (Properties)

Props in React refer to properties.

They are a mechanism to pass data from a parent component to a child component.

- **Read-Only:** Once a prop is passed to a component, it becomes read-only (or immutable) for that component. This means the child component should not modify the prop's value directly.
- **Data Flow:** Props allow for a one-way data flow, from parent to child, which makes the data flow in a React application predictable and easier to trace.

Props (Properties)

- **Type Checking:** With propTypes, you can ensure that the right type of data is being passed to the components. This helps in catching errors early.
- **Default Props:** React allows you to set default prop values using defaultProps.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
<Welcome name="React" />
```

Props - Activity

Building a User Profile Component

Create a user profile component that displays different user information using props.

Include :

Name
Age
Email

Props - Activity

userProfile.js

```
import React from 'react';

function UserProfile(props) {
  return (
    <div className="user-profile">
      <h2>User Profile</h2>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
      <p>Email: {props.email}</p>
    </div>
  );
}
export default UserProfile;
```

App.js

```
import React from 'react';
import UserProfile from './UserProfile';

function App() {
  return (
    <div className="App">
      <UserProfile
        name="John Doe"
        age={30}
        email="john@example.com"/>
    </div>
  );
}

export default App;
```

Hooks

A hook in React is a special kind of function that allows you to "hook into" React features from within function components

- useState
- useEffect
- useContext
- useReducer
- useCallback
- useMemo
- useRef
- useLayoutEffect

State

State refers to the local state of a component. It is data that dictates how a component renders and behaves. Unlike props, the state is changeable, but there's a proper way to change it.

Initialization: State is often initialized using the useState hook in functional components.

Changing State: In functional components, you use the setState function returned by the useState hook.

State

Rerender: Whenever the state changes, the component re-renders. It's one of the reasons why React components can be dynamic.

Local and Encapsulated: Each component has its own state. It's not accessible to any component other than the one that owns and sets it.

State - Example

```
import React, { useState } from 'react'

export default function State() {

  const [count, setCount] = useState(0);
  }
```

```
return (
 <div>
    <p>You clicked {count} times</p>
  <button onClick={() => setCount(count + 1)}>
    Click me
  </button>
 </div>
)
```

To Summarize

- Props allow parent components to pass data and functions down to child components.
- State allows components to maintain and modify their own data.
- Changes to either will trigger a rerender of the component.

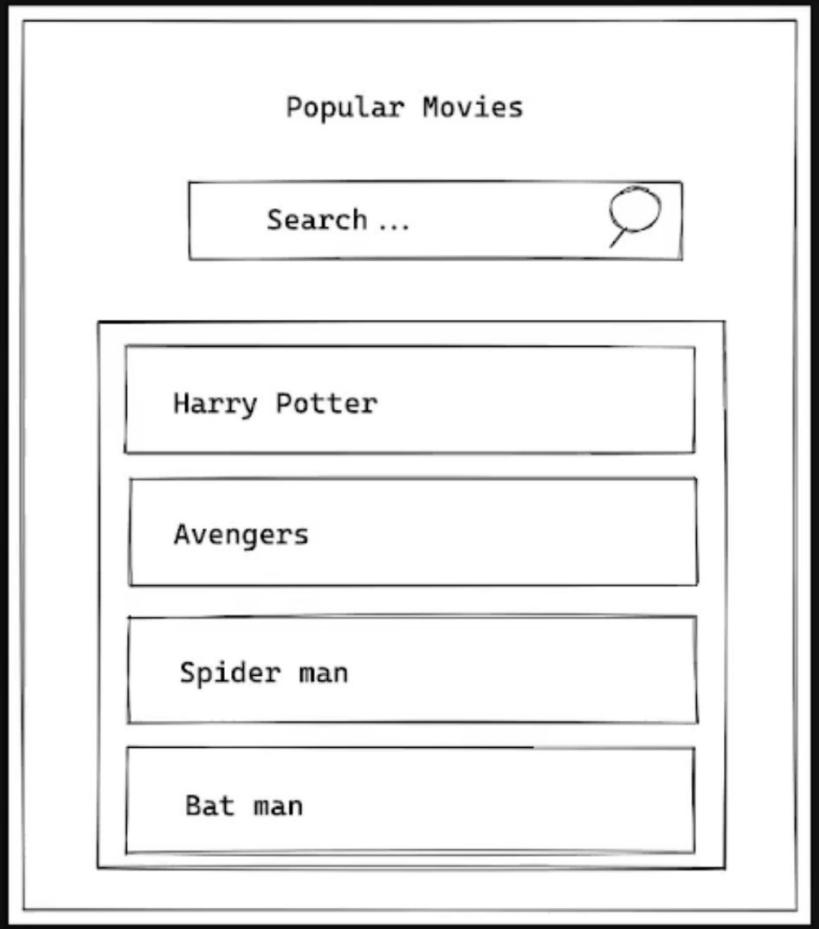
Final Project Details

You are suppose to develop a project at the end of the course and as the day 01 activity, you can finalize the idea and scope of your project. We expect you to upload a single page document with the idea and features along with a description to your project.

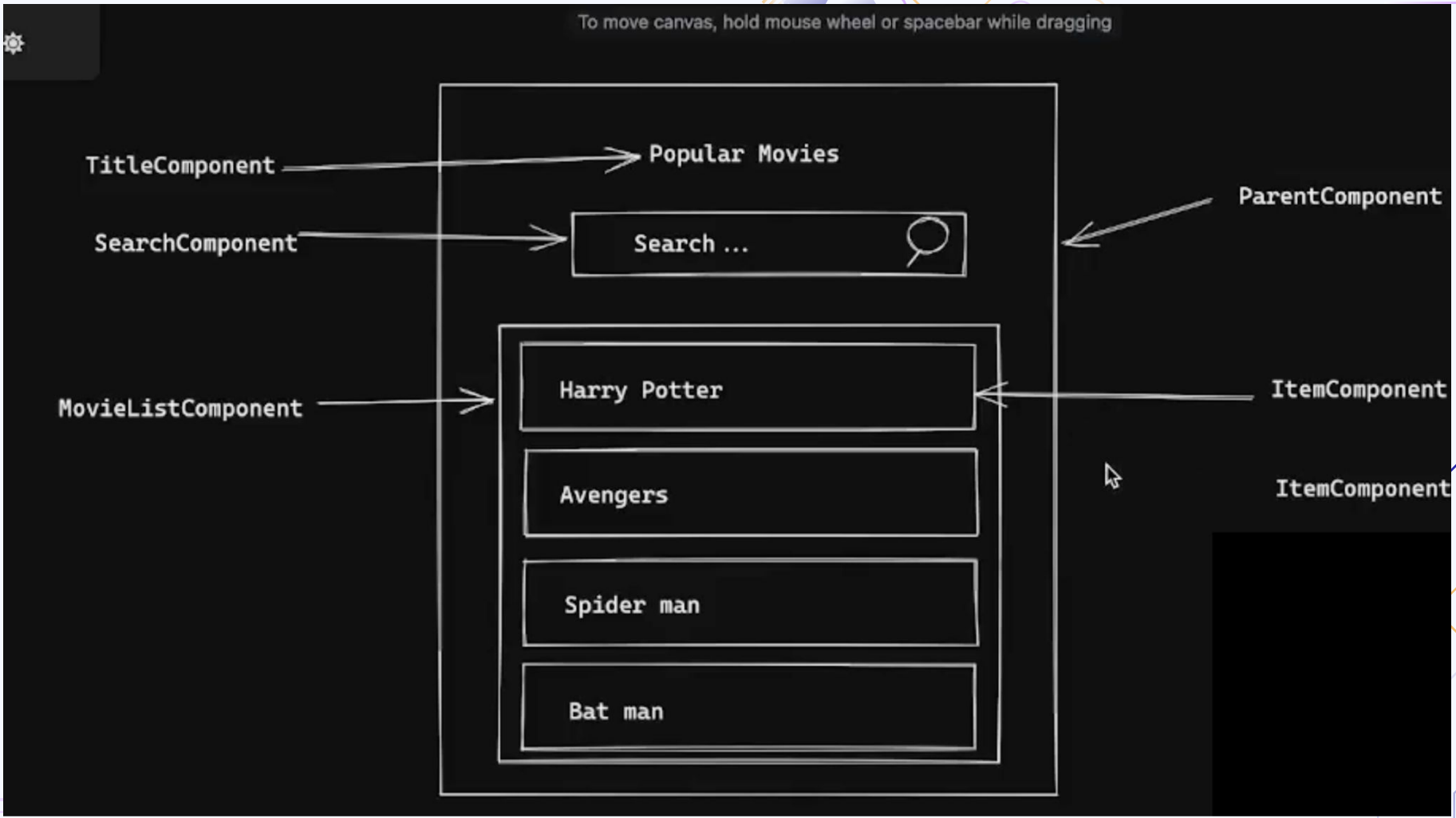
The minimum requirements of the project includes,

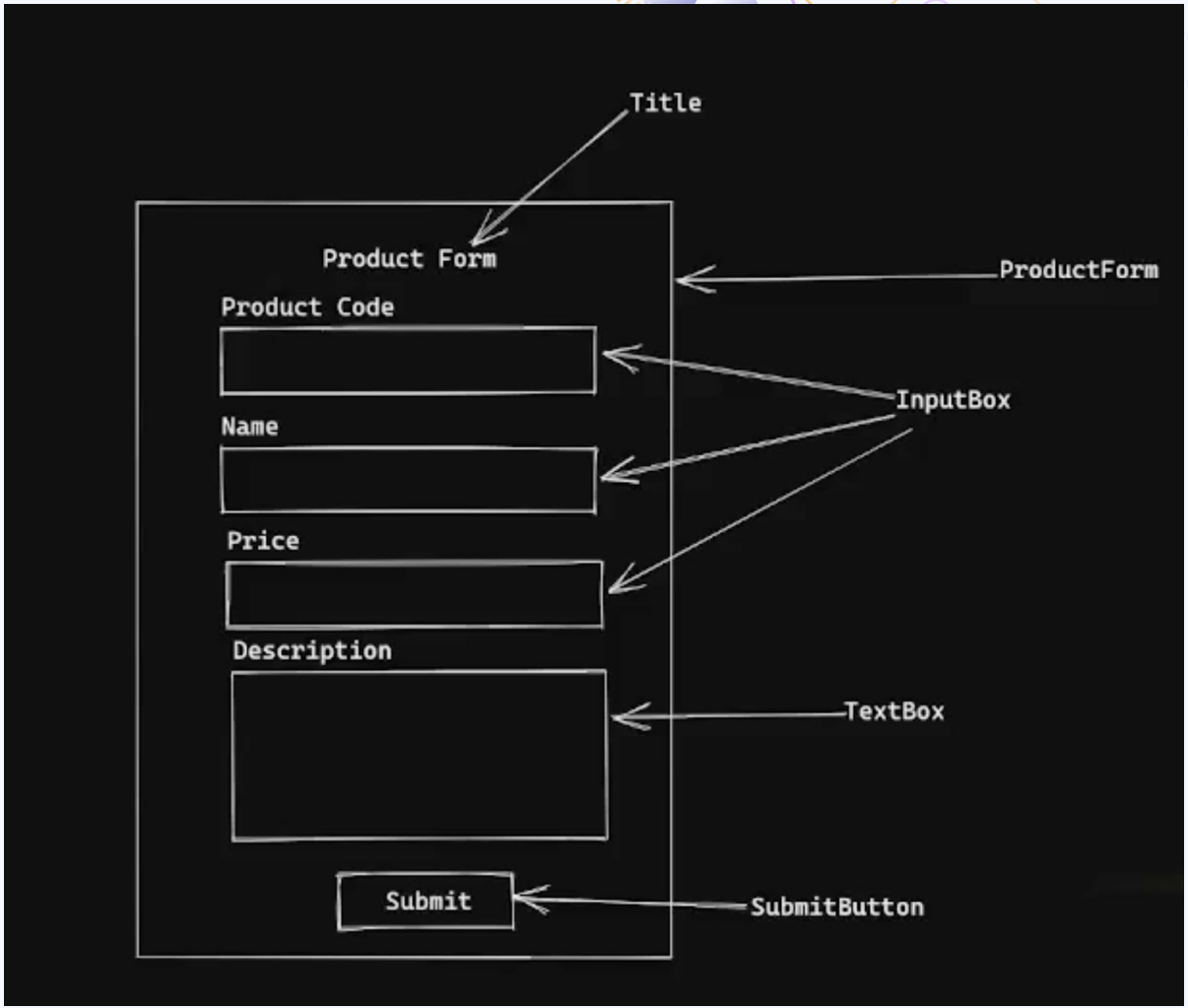
- A landing page
- Minimum of 2 user roles
- Registration and Login with authentication
- At least 1 CRUD operation for each user role (A CRUD operation consists of Create, Retrieve, Update and Delete)
- You should NOT use any pre-defined templates for this project and Any nice to have features are accepted and encouraged.

Understanding React Components


Popular Movies
Search ...
Harry Potter
Avengers
Spider man
Bat man


Product Form
Product Code
Name
Price
Description
Submit


To move canvas, hold mouse wheel or spacebar while dragging
TitleComponent
Popular Movies
ParentComponent
SearchComponent
Search ...
MovieListComponent
Harry Potter
ItemComponent
Avengers
ItemComponent
Spider man
Bat man


Title
Product Form
ProductForm
Product Code
InputBox
Name
Price
Description
TextBox
Submit
SubmitButton

Basic Script Tag setup

Create-react-app setup

What’s the Difference ?

How component render to the browser

- Our main container

```
<div id="root"></div>
```

- Our logic

```
<script type="text/babel">

    function MyButton(){
        return <button>My Button</button>
    }
    function Title() {
        return <h1>My Title</h1>
    }
    const element = document.getElementById("root");
    const root = ReactDOM.createRoot(element);
    root.render(<MyButton />)

</script>
```

How create react app handle that

- Public -> index.html .

```
<div id="root"></div>
```

- src -> index.js .

```
const root = ReactDOM.createRoot(document.getElementById('root'));
```

- src -> index.js .

```
root.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>
);
```

- Only component name is different
- So, where is the App component

```
import App from './App';
```

- src -> App.js .

```
function App() {
    return <SimpleCard/>
}
```

Now you can see

- Create react app is more organized
- Different components have different files
- Rendering part on another file (index.js)
- Have separate files loaded or imported (eg . css/svg)
- Conventions used (File name and Function name is same)
- Much more

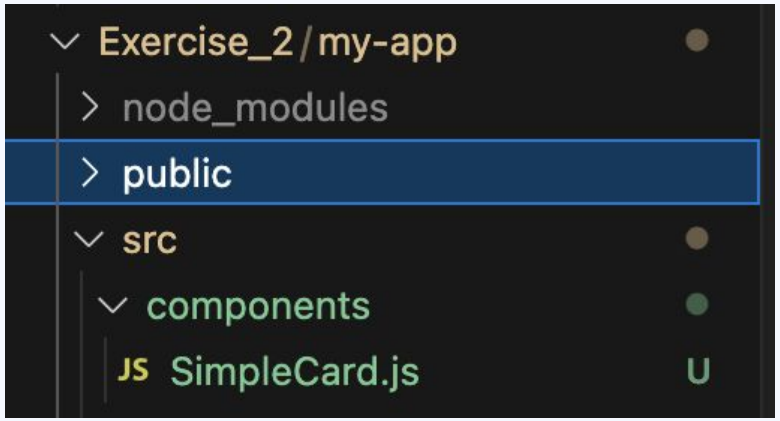
Let’s create a simple UI using React

1. Modify App.js

```
import './App.css';
function App() {
    return <h1>My App</h1>;
}
export default App;
```

2. Delete all content inside App.css

3. Create a react component (parent/for the whole card UI)



- Create a folder called “components” inside src(source) folder
- Then create a file called “SimpleCard.js” inside components folder

4. Edit SimpleCard.js & App.js

- Create a function called SimpleCard

```
function SimpleCard(){
    return <div>my simple card</div>
}
```

- Return my simple card

- Then we need to put our simple card inside App.js
  - Because App component is the main component of react app
  - When you go inside index.js you can see react only render App component

- To import a component, always we need to export that component first

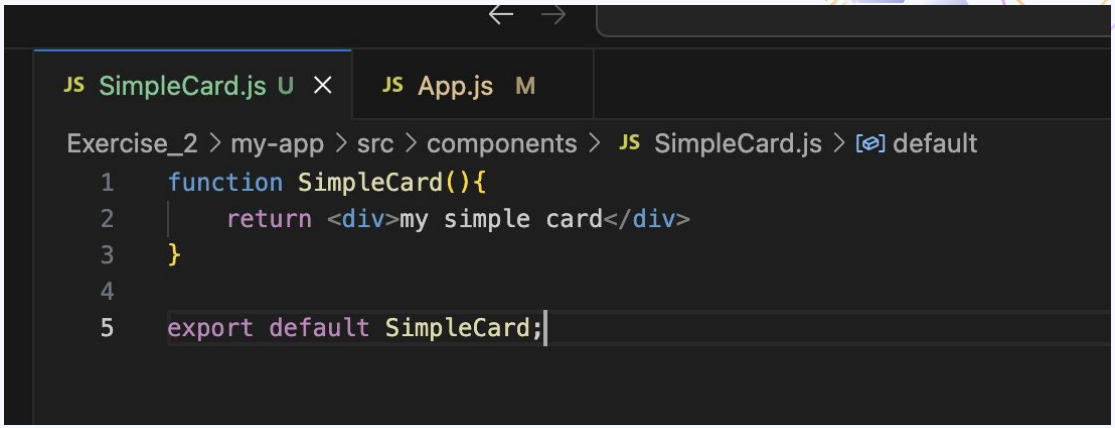
- **export default SimpleCard**

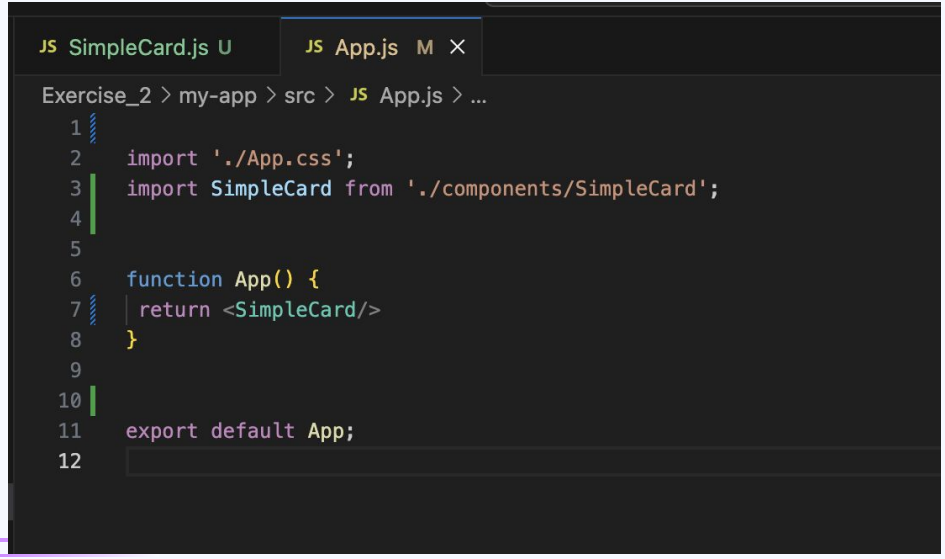
- Import simplecard to App.js
  - **import SimpleCard from './components/SimpleCard';**

- Then we have to return SimpleCard from App.js

```
function App() {
    return <SimpleCard/>
}
```

- Now run the code


SimpleCard.js U
App.js M
Exercise_2 > my-app > src > components > SimpleCard.js > default
function SimpleCard(){
return my simple card
}
export default SimpleCard;


SimpleCard.js U
App.js M
Exercise_2 > my-app > src > App.js > ...
import './App.css';
import SimpleCard from './components/SimpleCard';
function App() {
return <SimpleCard/>
}
export default App;

5. Now we'll create our simple card like UI

- Create a css file called SimpleCard.css
- First, we need to create the hierarchy inside SimpleCard.js
- Tip : we can't return two or more tags inside single line return statement
  - To do that use

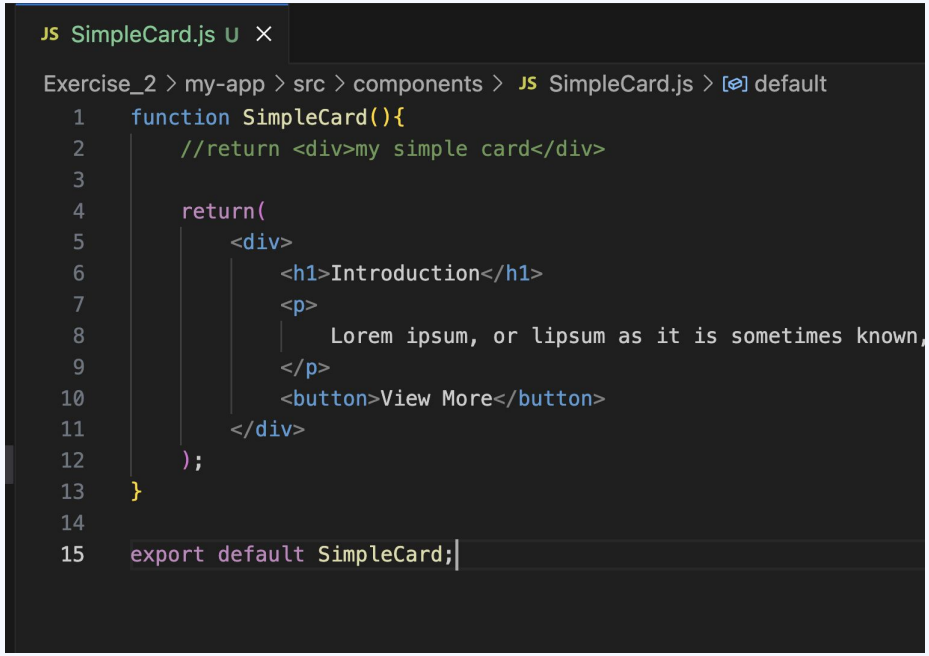
```
return(

);
```

  - Then need to have one parent div tag inside return

```
return(
    <div>
        .....add more tags here
    </div>
);
```

- Finalized hierarchy inside SimpleCard.js



- Now run the code

6. Now we can style the SimpleCard

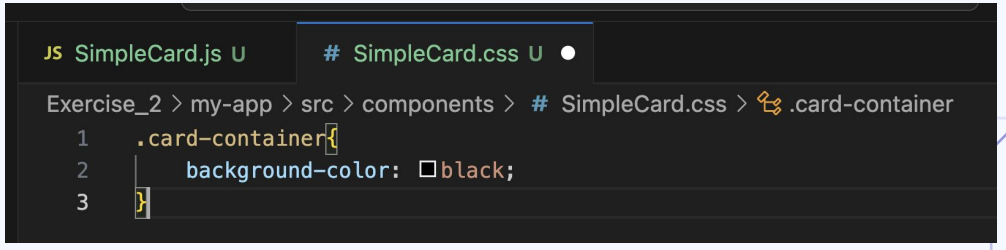
- First we need to import the SimpleCard.css file to SimpleCard.js
  import "./SimpleCard.css";

- Tip : In normal HTML we use “class=” when giving a name to a tag
  - However, in react we use “className=”
  - Its because react uses JSX

- Give className as card-containder for the parent div inside SimpleCard function
  - ** **

- Now go to SimpleCard.css and style

- We can give a background color to whole container to see the styles are working or not
- Save and check the output



7. Style the SimpleCard.css

- First go to index.css and chenge background color to black

```
body {
    background-color: black;
}
```

- Now go to SimpleCard.css and add the code
- Add paddings and boarder radius

```
# SimpleCard.css U ●
Exercise_2 > my-app > src > components > #
1  .card-container{
2      background-color: white;
3      border-radius: 15px;
4      padding-top: 10px;
5      padding-left: 25px;
6      padding-right: 25px;
7      padding-bottom: 35px;
8  }
```

- Now we'll add some awesome fonts
- Go to : https://fonts.google.com/
- Search for : poppins
- Select regular 400 and Bold 700 fonts
- Copy the link
- Add copied lines to
    - Public folder -> index.html -> Inside head tag

- Add **font-family: 'Poppins', sans-serif;** to css file
- Save and see

```
index.html ●
Exercise_2 > my-app > public > index.html > html > head > link
17    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
18    <!--
19      Notice the use of %PUBLIC_URL% in the tags above.
20      It will be replaced with the URL of the `public` folder during the build.
21      Only files inside the `public` folder can be referenced from the HTML.
22
23      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
24      work correctly both with client-side routing and a non-root public URL.
25      Learn how to configure a non-root public URL by running `npm run build`.
26    -->
27    <link rel="preconnect" href="https://fonts.googleapis.com">
28    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
29    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap"
30
31      <title>React App</title>
32    </head>
33    <body>
```

7. Style the SimpleCard.css cont…

- Change h1 to h2 inside SimpleCard.js
- Add following styles

SimpleCard.css U

Exercise_2 > my-app > src > components > # SimpleCard.c

```css
.card-container{
    background-color: white;
    border-radius: 15px;
    padding-top: 10px;
    padding-left: 25px;
    padding-right: 25px;
    padding-bottom: 35px;
    font-family: 'Poppins', sans-serif;
    width: 40%;
}

.card-container button{
    background-color: #8B5CF6;
    color: white;
    padding-top: 10px;
    padding-bottom: 10px;
    padding-left: 25px;
    padding-right: 25px;
    font-family: 'Poppins', sans-serif;
    border-radius: 10px;
    border: 0;
    font-size: 14px;

}
```

```css
.card-container p{
    color: #94A3B8;
    font-size: 15px;
    padding-bottom: 10px;
}

.card-container h2{
    color: #1E293B;
    font-size: 30px;
    margin: 0;
}
```

Create a simple navigation bar using multiple components

Thanks!